Evaluation of classifiers
MLPs

Milos Hauskrecht
milos@cs.pitt.edu
5329 Sennott Square

Evaluation

For any data set we use to test the model we can build a confusion matrix:
– Counts of examples with:
  – class label \( \omega_j \) that are classified with a label \( \alpha_i \)

<table>
<thead>
<tr>
<th>predict</th>
<th>( \omega = 1 )</th>
<th>( \omega = 0 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \alpha = 1 )</td>
<td>140</td>
<td>17</td>
</tr>
<tr>
<td>( \alpha = 0 )</td>
<td>20</td>
<td>54</td>
</tr>
</tbody>
</table>
## Evaluation

For any data set we use to test the model we can build a **confusion matrix**:

<table>
<thead>
<tr>
<th>predict</th>
<th>target</th>
<th>$\omega = 1$</th>
<th>$\omega = 0$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\alpha = 1$</td>
<td>140</td>
<td>17</td>
<td></td>
</tr>
<tr>
<td>$\alpha = 0$</td>
<td>20</td>
<td>54</td>
<td></td>
</tr>
</tbody>
</table>

### Agreement

$$\text{Error: } \frac{37}{231}$$

$$\text{Accuracy} = 1 - \text{Error} = \frac{194}{231}$$
Evaluation

- **Confusion matrix can be built for multi-way classification:**
  - Counts of examples with:
  - class label \( \omega_j \) that are classified with a label \( \alpha_i \)

<table>
<thead>
<tr>
<th></th>
<th>( \omega = 0 )</th>
<th>( \omega = 1 )</th>
<th>( \omega = 2 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \alpha = 0 )</td>
<td>140</td>
<td>20</td>
<td>22</td>
</tr>
<tr>
<td>( \alpha = 1 )</td>
<td>17</td>
<td>54</td>
<td>8</td>
</tr>
<tr>
<td>( \alpha = 2 )</td>
<td>12</td>
<td>4</td>
<td>76</td>
</tr>
</tbody>
</table>

agreement

Evaluation for binary classification

Entries in the confusion matrix for binary classification have names:

<table>
<thead>
<tr>
<th>predict</th>
<th>( \omega = 1 )</th>
<th>( \omega = 0 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \alpha = 1 )</td>
<td>TP</td>
<td>FP</td>
</tr>
<tr>
<td>( \alpha = 0 )</td>
<td>FN</td>
<td>TN</td>
</tr>
</tbody>
</table>

\( TP: \) True positive (hit)
\( FP: \) False positive (false alarm)
\( TN: \) True negative (correct rejection)
\( FN: \) False negative (a miss)
**Additional statistics**

- **Sensitivity (recall)**
  \[ SENS = \frac{TP}{TP + FN} \]

- **Specificity**
  \[ SPEC = \frac{TN}{TN + FP} \]

- **Positive predictive value (precision)**
  \[ PPT = \frac{TP}{TP + FP} \]

- **Negative predictive value**
  \[ NPV = \frac{TN}{TN + FN} \]

---

**Binary classification: additional statistics**

- **Confusion matrix**

<table>
<thead>
<tr>
<th>predict</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>140</td>
<td>10</td>
</tr>
<tr>
<td>0</td>
<td>20</td>
<td>180</td>
</tr>
</tbody>
</table>

  \[ PPV = \frac{140}{150} \]
  \[ NPV = \frac{180}{200} \]

  \[ SENS = \frac{140}{160} \]
  \[ SPEC = \frac{180}{190} \]

  **Row and column quantities:**
  - Sensitivity (SENS)
  - Specificity (SPEC)
  - Positive predictive value (PPV)
  - Negative predictive value (NPV)
Binary decisions: ROC

- Probabilities:
  - \( SENS \quad p(x > x^* \mid x \in \omega_2) \)
  - \( SPEC \quad p(x < x^* \mid x \in \omega_1) \)

Receiver Operating Characteristic (ROC)

- ROC curve plots:
  \( SN = p(x > x^* \mid x \in \omega_2) \)
  \( 1 - SP = p(x > x^* \mid x \in \omega_1) \)
  for different \( x^* \)
ROC curve

Case 1
Case 2
Case 3

$p(x > x^* | x \in \omega_2) = \cdots$

Receiver operating characteristic

- **ROC**
  - shows the discriminability between the two classes under different decision biases
- **Decision bias**
  - can be changed using different loss function
Zero-one loss function

- **Misclassification error**
  - Based on the zero-one loss function
    - Any misclassified example counts as 1
    - Correctly classified example counts as 0

<table>
<thead>
<tr>
<th></th>
<th>$\omega = 0$</th>
<th>$\omega = 1$</th>
<th>$\omega = 2$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\alpha = 0$</td>
<td>140</td>
<td>20</td>
<td>22</td>
</tr>
<tr>
<td>$\alpha = 1$</td>
<td>17</td>
<td>54</td>
<td>8</td>
</tr>
<tr>
<td>$\alpha = 2$</td>
<td>12</td>
<td>4</td>
<td>76</td>
</tr>
</tbody>
</table>

agreement

General loss function

- **Error function based on a more general loss function**
  - Different misclassifications have different weight (loss)
  - $\alpha$, our choice
  - $\omega_j$, true label
  - $\lambda(\alpha_i \mid \omega_j)$ loss for classification

Example:

<table>
<thead>
<tr>
<th>$\lambda(\alpha_i \mid \omega_j)$</th>
<th>$\omega = 0$</th>
<th>$\omega = 1$</th>
<th>$\omega = 2$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\alpha = 0$</td>
<td>0</td>
<td>1</td>
<td>5</td>
</tr>
<tr>
<td>$\alpha = 1$</td>
<td>3</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>$\alpha = 2$</td>
<td>3</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

CS 2750 Machine Learning
Bayesian decision theory

• More general loss function
  – Different misclassifications have different weight (loss)
    \( \lambda (\alpha_i \mid \omega_j) \)

• Expected loss for the classification choice \( \alpha_i \)
  \[ R(\alpha_i \mid x) = \sum_j \lambda (\alpha_i \mid \omega_j) P(y = \omega_j \mid x) \]
  – Also called conditional risk

• Decision rule: \( \alpha(x) \)
  – Chooses label (action) according to the input

• The optimal decision rule
  \[ \alpha^*(x) = \arg \min_{\alpha_j} \sum_j \lambda (\alpha_i \mid \omega_j) P(y = \omega_j \mid x) \]

How to modify classifiers to handle different loss?

• Discriminative models:
  – Directly optimize the parameters according to the new loss function

• Generative models:
  – Learn probabilities as before
  – Decisions about classes are biased to minimize the empirical loss (as seen above)
Calculating the loss for data

- **Confusion matrix:**
  - Counts of examples with:
  - class label $\omega_j$ that are classified with a label $\alpha_i$

<table>
<thead>
<tr>
<th></th>
<th>$\omega = 0$</th>
<th>$\omega = 1$</th>
<th>$\omega = 2$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\alpha = 0$</td>
<td>140</td>
<td>20</td>
<td>22</td>
</tr>
<tr>
<td>$\alpha = 1$</td>
<td>17</td>
<td>54</td>
<td>8</td>
</tr>
<tr>
<td>$\alpha = 2$</td>
<td>12</td>
<td>4</td>
<td>76</td>
</tr>
</tbody>
</table>

\[ \text{agreement} \]

- **Loss**

\[
\frac{1}{N} \sum_i \sum_j \lambda (\alpha_i \mid \omega_j) N (\alpha_j \mid \omega_j)
\]

Multilayer neural networks
Linear units

Linear regression

\[ f(\mathbf{x}) = w_0 + \sum_{j=1}^{d} w_j x_j \]

Logistic regression

\[ f(\mathbf{x}) = p(y = 1 | \mathbf{x}, \mathbf{w}) = g(w_0 + \sum_{j=1}^{d} w_j x_j) \]

On-line gradient update:

\[ w_0 \leftarrow w_0 + \alpha (y - f(\mathbf{x})) \]
\[ w_j \leftarrow w_j + \alpha (y - f(\mathbf{x})) x_j \]

Limitations of basic linear units

Linear regression

\[ f(\mathbf{x}) = w_0 + \sum_{j=1}^{d} w_j x_j \]

Function linear in inputs !!

Logistic regression

\[ f(\mathbf{x}) = p(y = 1 | \mathbf{x}, \mathbf{w}) = g(w_0 + \sum_{j=1}^{d} w_j x_j) \]

Linear decision boundary!!
Regression with the quadratic model.

**Limitation:** linear hyperplane only
- a non-linear surface can be better

Classification with the linear model.

**Logistic regression model defines a linear decision boundary**
- Example: 2 classes (blue and red points)
Linear decision boundary

- logistic regression model is not optimal, but not that bad

When logistic regression fails?

- Example in which the logistic regression model fails
Limitations of linear units.

- Logistic regression does not work for \textit{parity functions} - no linear decision boundary exists

\[ f(x) = w_0 + \sum_{j=1}^{m} w_j \phi_j(x) \]

\[ f(x) = g(w_0 + \sum_{j=1}^{m} w_j \phi_j(x)) \]

\[ \phi_j(x) \quad - \text{an arbitrary function of } x \]

Solution: a model of a non-linear decision boundary

Extensions of simple linear units

- use \textit{feature (basis) functions} to model \textit{nonlinearities}

Linear regression \hspace{1cm} Logistic regression
\[ f(x) = w_0 + \sum_{j=1}^{m} w_j \phi_j(x) \hspace{1cm} f(x) = g(w_0 + \sum_{j=1}^{m} w_j \phi_j(x)) \]

\[ \phi_j(x) \quad - \text{an arbitrary function of } x \]
Learning with extended linear units

**Feature (basis) functions** model **nonlinearities**

**Linear regression**
\[ f(x) = w_0 + \sum_{j=1}^{m} w_j \phi_j(x) \]

**Logistic regression**
\[ f(x) = g\left(w_0 + \sum_{j=1}^{m} w_j \phi_j(x)\right) \]

**Important property:**
- The same problem as learning of the weights for linear units, the input has changed— but the weights are linear in the new input

**Problem:** too many weights to learn

---

Multi-layered neural networks

- Alternative way to introduce nonlinearities to regression/classification models
- **Idea:** Cascade several simple neural models with logistic units. Much like neuron connections.

---
Multilayer neural network

Also called a **multilayer perceptron (MLP)**
Cascades multiple logistic regression units

**Example:** (2 layer) classifier with non-linear decision boundaries

\[
\sum_{1} z_1(1) \rightarrow w_{0,1}(2) \rightarrow z_2(1) \rightarrow w_{2,1}(2) \rightarrow p(y = 1 | x)
\]

- Models **non-linearities through logistic regression units**
- Can be applied to both **regression and binary classification problems**

Input layer | Hidden layer | Output layer
---|---|---
\[1, x_1, x_2, \ldots, x_d\] | \[z_1(1), z_2(1), \ldots, z_i(1)\] | \[p(y = 1 | x)\]

**regression** \[f(x) = f(x, w)\]

**classification** \[f(x) = p(y = 1 | x, w)\]
Multilayer neural network

- Non-linearities are modeled using multiple hidden logistic regression units (organized in layers)
- The output layer determines whether it is a regression or a binary classification problem

\[ f(x) = f(x, w) \]

\[ \int f(x) = p(y = 1 | x, w) \]